Comparative Analysis of Time Series Databases in the Context of Edge Computing for Low Power Sensor Networks

Piotr Grzesik[1] and Dariusz Mrozek[1]

Department of Applied Informatics, Silesian University of Technology
ul. Akademicka 16, 44-100 Gliwice, Poland
dariusz.mrozek@polsl.pl

Abstract. Selection of an appropriate database system for edge IoT devices is one of the essential elements that determine efficient edge-based data analysis in low power wireless sensor networks. This paper presents a comparative analysis of time series databases in the context of edge computing for IoT and Smart Systems. The research focuses on the performance comparison between three time-series databases: TimescaleDB, InfluxDB, Riak TS, as well as two relational databases, PostgreSQL and SQLite. All selected solutions were tested while being deployed on a single-board computer, Raspberry Pi. For each of them, the database schema was designed, based on a data model representing sensor readings and their corresponding timestamps. For performance testing, we developed a small application that was able to simulate insertion and querying operations. The results of the experiments showed that for presented scenarios of reading data, PostgreSQL and InfluxDB emerged as the most performing solutions. For tested insertion scenarios, PostgreSQL turned out to be the fastest. Carried out experiments also proved that low-cost, single-board computers such as Raspberry Pi can be used as small-scale data aggregation nodes on edge device in low power wireless sensor networks, that often serve as a base for IoT-based smart systems.

Keywords: time series, PostgreSQL, TimescaleDB, InfluxDB, edge computing, edge analytics, Raspberry Pi, Riak TS, SQLite

1 Introduction

In the recent years we have been observing IoT systems being applied for multiple use cases such as water monitoring[20], air quality monitoring [24], and health monitoring [25], generating a massive amount of data that is being sent to the cloud for storing and further processing. This is becoming a more significant challenge due to the need for sending the data over the Internet. Due to that, a new computing paradigm called edge computing started to emerge [28]. The main idea behind edge computing is to move data processing from the cloud to the devices that are closer to the source of data in order to reduce the volume of data that needs to be send to the cloud, improve reaction time to the changing state of

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50426-7_28

the system, provide resilience and prevent data loss in situations where Internet connection is not reliable or even not available most of the time. To achieve that, edge computing devices need to be able to ingest data from sensors, analyze them, aggregate metrics, and send them to the cloud for further processing if required. For example, while collecting and processing environmental data on air quality, the edge device can be responsible for aggregating data and computing Air Quality Index (AQI) [22], instead of sending raw sensor readings to the environmental monitoring center. In systems with multiple sensors generating data at a fast rate, efficient storage and analytical system running on edge device becomes a crucial part. Due to the time-series nature of sensor data, dedicated time series databases seem like a natural fit for this type of workload. This paper aims to evaluate several time series databases in the context of using them in edge computing, low-cost, constrained device in form of Raspberry Pi that is processing data from environmental sensors. The paper is organized as follows. In section 2, we review the related works. In section 3, we describe databases selected for comparison. Section 4 describes testing environment, used data model as well as testing methodology. Section 5 contains a description of the performance experiments that we carried out. Finally, section 6 concludes the results of the paper.

2 Related Works

In the literature, there is a few research concerning the comparison of various time-series databases. In the paper [27], Tulasi Priyanka Sanaboyina compared two time-series databases, InfluxDB and OpenTSDB, based on the energy consumption of the physical servers on which the databases are running under several reading and writing scenarios. The author concludes the research with claims that InfluxDB consumes less energy than OpenTSDB in comparable situations.

Bader et al. [17] focused on open source time-series databases, examined 83 different solutions during their research, and focused on the comparison of twelve selected databases, including InfluxDB, PostgreSQL and OpenTSDB among others. All selected solutions were compared based on their scalability, supported functions, granularity, available interfaces, and extensions as well as licensing and support.

In his research [21], Goldschmidt et al. benchmarked three open-source time-series databases, OpenTSDB, KariosDB and Databus in the cloud environment with up to 36 nodes in the context of industrial workloads. The main objective of the research was to evaluate selected databases to determine their scalability and reliability features. Out of the three technologies, KairosDB emerged as the one that meets the initial hypotheses about scalability and reliability.

Wlodarczyk, in his article [29], provides an overview and comparison of four offerings, Chukwa, OpenTSDB, TempoDB, and Squwk. The analysis focused on feature differences between selected technologies, without any performance benchmarks. The author identified OpenTSDB as a most popular choice for the time series storage.

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50426-7_28

Pungilă et al. [26] compared the databases to use them in the system that stores large volumes of sensor data from smart meters. During the research, they compared three relational databases, SQLite3, MySQL, PostgreSQL, one time-series database, IBM Informix with DataBlade module, as well as three NoSQL databases, MonetDB, Hypertable and Oracle BerkeleyDB. During the experiments, it was determined that Hypertable offers the most significant number of insert operations per second, but is slower when it comes to scanning operations. The authors suggested that BerkeleyDB offers a compromise when there is a need for a workload that has a balanced number of both insert and scan operations.

Fadhel et al. presented research [20] concerning the evaluation of the databases for a low-cost water quality sensing system. Authors identified InfluxDB as the most suitable solution, listing the ease of installation and maintenance, support for multiple interface formats, and HTTP GUI as the deciding factors. In the second part of the research, they conducted performance experiments and determined that InfluxDB can handle the load from 450 sensors.

In his article [23], Kiefer provided a performance comparison between PostgreSQL and TimescaleDB for storage and analytics of large scale, time-series data. The author presented that at the scale of millions of rows, TimescaleDB offers up to 20x higher ingest rates than PostgreSQL, at the same time offering time-based queries to be even 14,000x faster. The author also mentions that for simple queries, e.g., indexed lookups, TimescaleDB will be slower than PostgreSQL due to more considerable planning time.

Boule, in his work [19], described a performance comparison for insert and read operations between InfluxDB and TimescaleDB. It is based on a simulated dataset of metrics for a fleet of trucks. According to results obtained during the experiments, TimescaleDB offers a better read performance than InfluxDB in tested scenarios.

Based on the above, it can be concluded that most of the current research focuses on on the use of time-series databases for large-scale systems, running in cloud environments. One exception to that is the research [20], where authors evaluate several databases in the context of a low-cost system; however, presenting performance tests only for one of them, InfluxDB. In contrast to the mentioned works, this paper focuses on the comparison of the performance of several database systems for storing sensor data at the edge devices that have limited storage and compute capabilities.

3 Time-Series Databases

Time series database (TSDB) is a database type designed and optimized to handle timestamped or time-series data, which is characterized by a low number of relationships between data and temporal ordering of records. Most of the time series workloads consist of a high number of insert operations, often in batches. Query patterns include some forms of aggregation over time. It is also important to note that in such workloads, data usually does not require

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50426-7_28

updating after being inserted. To accommodate these requirements, time-series databases store data in the form of events, metrics, or measurements, typically numerical, together with their corresponding timestamps and additional labels or tags. Data is very often chunked, based on timestamp, which in turn allows for fast and efficient time-based queries and aggregations. Most TSDBs offer advanced data processing capabilities such as window functions, automatic aggregation functions, time bucketing, and advanced data retention policies. There are currently a few approaches to building a time-series database. Some of them, like OpenTSDB or TimescaleDB, depend on already existing databases, such as HBase or PostgreSQL, respectively, while others are standalone, independent systems such as InfluxDB. In recent years, according to DB Engine ranking, as seen in Fig.1, the growth rate of the popularity of time series databases is the highest out of all classified database types. For the experiments, databases were selected based on their popularity, offered aggregation functionalities, support for ARM architecture, SQL or SQL-like query language support as well as on their availability without commercial license.

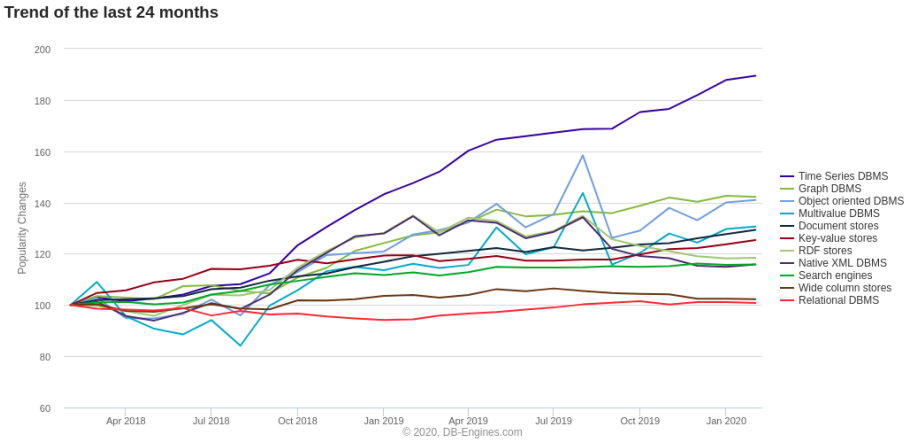


Fig. 1: Growth trend of various types of databases in the last 24 months according to DB-Engines.com[2]

3.1 TimescaleDB

TimescaleDB is an open-source, time-series database, written in C programming language and is distributed as an extension of the relational database, PostgreSQL. It is developed by Timescale Inc., which also offers enterprise support and cloud hosting in the form of Timescale Cloud offering. TimescaleDB is optimized for fast ingest and complex queries [14]. Thanks to the support

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50426-7_28

for all SQL operations available in PostgreSQL, it can be used as a drop-in replacement of a traditional relational database, while also offering significant performance improvements for storing and processing time-series data. By taking advantage of automatic space-time partitioning, it enables horizontal scaling, which in turn can further improve the ingestion capabilities of the system. It stores data in structures called hypertables, which serve as an abstraction for a single, continuous table. Internally, TimescaleDB splits hypertables into chunks that correspond to a specific time interval and partition keys. Chunks are implemented by using regular PostgreSQL tables [16]. Thanks to being an extension of PostgreSQL DBMS, it supports the same client libraries that support PostgreSQL. According to the DB Engines ranking [15], it is the 8th most popular time-series database.

3.2 InfluxDB

InfluxDB is an open-source, time-series database, written in Go programming language, developed and maintained by InfluxDB Inc., which also offers enterprise support and a cloud-hosted version of the database. Internally, it uses a custom-build storage engine called Time-Structured Merge (TSM) Tree, which is optimized for time series data. It has no external dependencies, is distributed as a single binary, which in turn allows for easy deployment process on all major operating systems and platforms. InfluxDB supports InfluxQL, which is a custom, SQL-like query language with support for aggregation functions over time series data. It supports advanced data retention policies as well as continuous queries, which allow for automatic computations of aggregate data to speed up frequently used queries [5]. It uses shards to partition data and organizes them into shards groups, based on the retention policy and timestamps. InfluxDB is also a part of TICK stack [4], which is a data processing platform that consists of a time-series database in form of InfluxDB, Kapacitor, which is a real-time streaming data processing engine, Telegraf, the data collection agent and Chronograf, a graphical user interface to the platform. Client libraries in the programming languages like Go, Python, Java, Ruby, and others are available, as well as command-line client "influx". According to DB Engines ranking [3], it is the most popular time-series database management system.

3.3 Riak TS

Riak TS is an open-source, distributed NoSQL database, optimized for the time series data and built on top of Riak KV database [9], created and maintained by Basho Technologies. Riak TS is written in Erlang programming language, supports masterless, multi-node architecture to ensure resiliency to network and hardware failures. This type of architecture also allows for efficient scalability with near-linear performance increase[10]. It supports a SQL-like query language with aggregation operations over time series data. It offers both HTTP and PBC APIs as well as dedicated client libraries in Java, Python, Ruby, Erlang, and Node.js. Besides, it has a native Apache Spark [1] connector for the in-memory

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50426-7_28

analytics. According to DB Engines ranking [11], it is the 15th most popular time-series database.

3.4 PostgreSQL

PostgreSQL is an open-source relational database management system written in C language and currently maintained by PostgreSQL Global Development Group. PostgreSQL runs on all major operating systems, is ACID [30] compliant and supports various extensions, namely TimescaleDB. It supports a major part of the SQL standard and offers many features, including but not limited to, triggers, views, transactions, streaming replication. It uses multi-version concurrency control, MVCC [18]. In addition to being a relational database, it also offers support for storing and querying document data thanks to JSON, JSONB, XML, and Key-value data types [6]. There are client libraries available in programming languages like Python, C, C++, Java, Go, Erlang, Rust, and others. According to DB Engines ranking [7], it is the 4th most popular database overall. It does not offer any dedicated support and optimizations for time-series data.

3.5 SQLite

SQLite is an open-source relational database, written in C language. The SQLite source code is currently available in the public domain. It is a lightweight, single file, and unlike most databases, it is implemented only as a library and does not require a separate server process. SQLite provides all functionalities directly by the function calls. Its simplicity makes it one of the most widely used databases, especially popular in embedded systems. SQLite has a full-featured SQL standard implementation with support for functionalities such as triggers, views, indexes, and many more [12]. Similar to PostgreSQL, it does not offer any specific support for time series data. Besides, it does not provide a data type for storing time, and it requires users to save it as numerical timestamps or strings. According to DB Engines ranking [13], it is the 7th most popular relational database and 10th most popular database overall.

4 Testing Environment and Data Model

The testing environment was based on a 6LoWPAN sensor network that is a part of the environment monitoring system, which consists of a group of the edge router device that additionally serves as a database and analytical engine. It is also responsible for sending aggregated metrics to the analytic system in the cloud for further processing. Another part of the network is composed of ten sensor nodes that are sending measurements such as air quality and weather condition metrics to the edge router device. Fig. 2 presents the network diagram of the described system.

In this research, we focused on performance evaluation of the edge database functionality of the presented system. To simplify the testing environment and

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50426-7_28

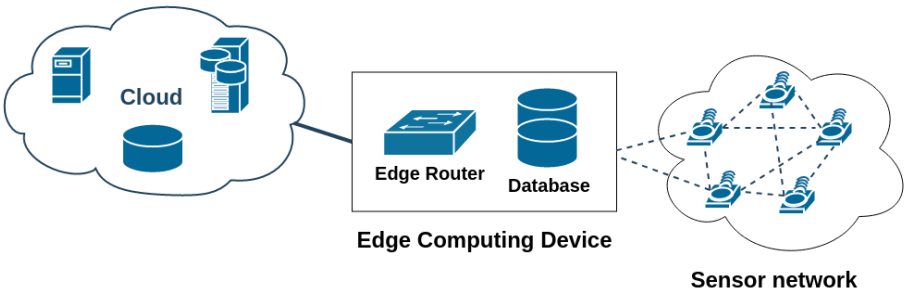


Fig. 2: Network diagram of the edge computing system.

allow for running tests multiple times in a reasonable amount of time, we developed a small Python application to serve as a generator of sensor readings instead of using data generated by the physical network. As an edge device we decided to use a Raspberry Pi single-board computer, with the following specification [8]:

- CPU - Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz
- Memory - 4GB LPDDR4-3200 SDRAM
- Storage - SDHC card (16 GB, class 10)
- OS - Raspbian GNU/Linux 10 (buster) with kernel version 4.19.50-v7l+

Table 1: Data model used for the performance experiments

Value	**Type**
Temperature	Float
Pressure	Float
Humidity	Float
PM2.5	Float
PM10	Float
NO_2	Float
Sensor ID	String
Location	String
Time	Timestamp (Integer)

4.1 Data model

Each data point sent by the sensor consists of air quality metrics in the form of NO_2 and dust particle size metrics – PM2.5 and PM10. Besides, it also carries information about weather conditions such as ambient temperature, pressure,

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50426-7_28

and humidity. Each reading is timestamped and tagged with the location of the sensor and the unique sensor identifier. Table 1 shows the structure of a single data point with corresponding data types. For the experiments, we generated data from 10 simulated sensors, where each sensor sends reading every 15 seconds over 24 hours. It resulted in 28,800 data points used for performance testing.

4.2 Testing methodology

For testing, a small Python application was developed separately for each of the selected databases. The application was responsible for reading simulated time-series data, inserting that data into the database and reading the data back from the database, while measuring the time it took to execute all of the described operations. Table 2 presents the list of the databases along with their corresponding client libraries. It also shows versions of the software used during the experiments.

Table 2: Database and client library versions

Database	**Database version**	**Client library**	**Client library version**
TimescaleDB	1.5.1	psycopg2	2.8.4
InfluxDB	1.7.9	influxdb	5.2.3
Riak TS	1.5.2	riak	2.7.0
PostgreSQL	11.5	psycopg2	2.8.4
SQLite	3.27.2	sqlite3	2.6.0

5 Performance Experiments

To evaluate the insertion and querying performance, we conducted several experiments. Firstly, we ran the test to assess the writing capabilities of all selected databases by simulating the insertion of data points in two ways: one-by-one and in batches of 10 points. The reason for that was to accommodate the fact that databases can offer better performance for batch insertions, and it is possible to buffer data before saving it to the database. In this step, for each database, we ran the simulation 50 times (except for SQLite where simulations were run 20 times due to relatively long simulation time). Secondly, we ran the experiments to evaluate the query performance of all selected solutions in three scenarios. In the first scenario, we evaluated a query for average temperature in the chosen period, grouped by location. In the second query, we tested a query for minimum and maximum values of NO_2, PM2.5, and PM10 in the selected period, once again grouped by location. In the last, third scenario, we evaluated the performance of a query that counts data points grouped by sensor ID in the selected period for which NO_2 was larger than selected value and location was equal to a specific one. Each query was executed 5000 times. The query scenarios

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50426-7_28

were selected in order to test the performance of the databases for most common aggregation queries that can be used in scenarios where the analysis has to be performed directly on the edge device or when the data needs to be aggregated before sending to the cloud in order to reduce the volume of transferred data.

5.1 Insertion

In the first simulation, we evaluated the insertion performance in two different scenarios. Fig. 3 presents the obtained results in the form of the average number of data points inserted per second in both scenarios. For one-by-one insertion, we observe PostgreSQL and TimescaleDB as the best performing solutions, with 260 and 230 points inserted per second, respectively. Next is Riak TS with 191 points, followed by InfluxDB with 54 points per second. On the other side of the spectrum is SQLite, with only 8 points per second inserted on average. In the second scenario, with batch insertions of 10-point batches, we observed a general trend of higher ingestion rates for all databases in comparison to single point writes with InfluxDB being 8.65 times faster, both PostgreSQL and SQLite improving 6.74 times, TimescaleDB improving 5.15 times and Riak TS noting the smallest relative increase by 2.45 times. We observed similar results as for one-by-one insertion, with PostgreSQL being the most performant database in the tested scenario with 1,756 data points ingested per second, followed by TimescaleDB with 1,187 points, InfluxDB with 474 points and Riak TS with 468 points. Once again, SQLite recorded the worst performance, with only 55 points ingested per second.

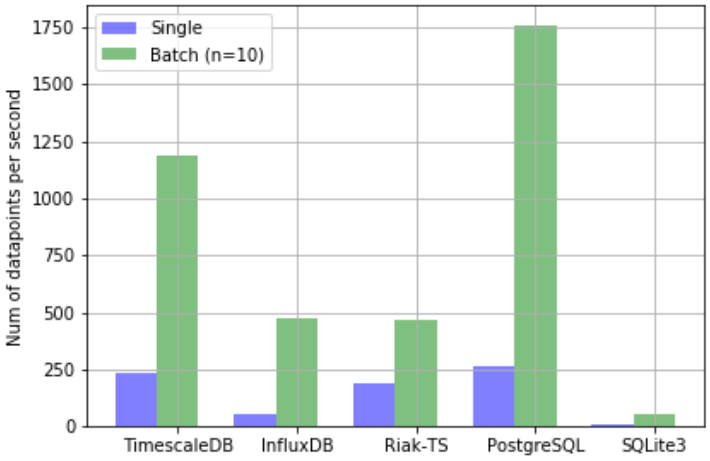


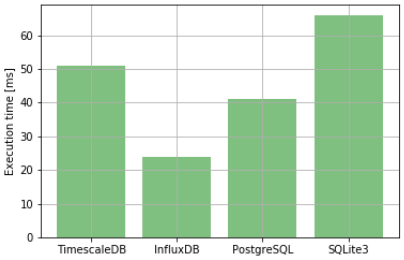
Fig. 3: Number of data points ingested per second for each tested database.

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50426-7_28

5.2 Querying

In the following experiments, we tested the reading performance for three different queries. Results are presented in the form of the average query execution time in milliseconds for each database. Due to the fact that execution for Riak TS was in all cases 20-40 times slower than for all other solutions, the results for Riak TS were removed from the further comparison to improve the readability of the presented charts. Fig. 4 shows both the query used in the first scenario as well as the obtained results. In this scenario, InfluxDB emerged as the fastest solution with average query execution time of 24 milliseconds, followed by PostgreSQL and TimescaleDB with 41 and 52 milliseconds, respectively. SQLite was the slowest, recording average query execution time of 66 milliseconds.

```
SELECT
  location,
  AVG("temperature")
from
  "conditions"
WHERE
  time >= '2019-12-20T03:00:00Z'
  AND time <= '2019-12-20T15:00:00Z'
GROUP BY
  "location"
```



(a) Tested aggregation query (b) Average query execution time

Fig. 4: Query and test results for the first querying scenario.

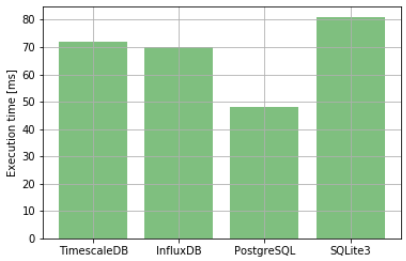
Next, a comparison was made for the results obtained during the evaluation of second query computing minimum and maximum aggregations of air quality metrics. The recorded results and queries are shown in Fig. 5. In this example, PostgreSQL turned out to be the fastest solution with average query execution time of 48 milliseconds, next was InfluxDB with 70 milliseconds and TimescaleDB with 72 milliseconds. Tested query took the longest time to execute on SQLite, taking on average 81 milliseconds. We can observe a general trend of increased query execution time with more aggregations performed in comparison to the first testing scenario.

The last experiment was performed for the third tested query, evaluating the number of times the NO_2 was higher than the predefined threshold. Fig. 6 presents the query used and the results obtained during that simulation. Once

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50426-7_28

```
SELECT
  location,
  MIN("pm_2_5"),
  MAX("pm_2_5"),
  MIN("pm_10"),
  MAX("pm_10"),
  MIN("no2"),
  MAX("no2")
from
  "conditions"
WHERE
  time >= '2019-12-20T03:00:00Z'
  AND time <= '2019-12-20T15:00:00Z'
GROUP BY
  "location"
```

(a) Tested aggregation query



(b) Average query execution time

Fig. 5: Query and test results for the second querying scenario.

again, PostgreSQL was the fastest solution with an average query execution time of 15 milliseconds, followed by InfluxDB with 29 milliseconds. The two slowest databases were TimescaleDB and SQLite, with 39 and 40 milliseconds per execution on average.

5.3 Results Summary

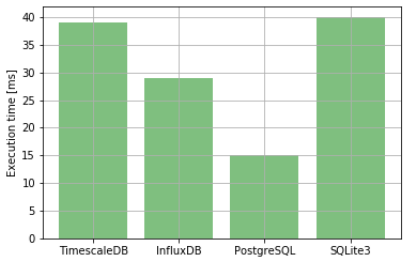
Considering results for all presented simulations, we can observe that in almost all cases, PostgreSQL is the best performing solution for the evaluated workloads, except for InfluxDB, which turned out to be faster for the first aggregation query. It was validated that batching data points for insertion causes performance gains, as high as 8.65 times more data points ingested per second for InfluxDB. With the exception of Riak TS, all databases executed tested queries on average in less than 80 milliseconds, and the relative differences in performance for queries are not as high as in the case of insertion.

6 Concluding Remarks

The selection of a proper storage system with declarative querying capabilities is an essential element of building efficient systems with edge-based analytics. This research aimed to compare the performance of several databases in the context of edge computing in wireless sensor networks for IoT-based smart systems. We believe that experiments and analysis of the results presented in the paper complement the performance evaluation of InfluxDB presented in [20] by showcasing performance results for multiple databases and can serve as a reference

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50426-7_28

```
SELECT
  sensor_id,
  Count("no2")
from
  "conditions"
WHERE
  "no2" > 8
  AND "location" = 'outdoor'
  AND time >= '2019-12-20T03:00:00Z'
  AND time <= '2019-12-20T15:00:00Z'
GROUP BY
  "sensor_id"
```



(a) Tested aggregation query (b) Average query execution time

Fig. 6: Query and test results for the third querying scenario.

when selecting an appropriate database for low-cost, edge analytics applications. As it turned out, for a smaller scale, it might make sense to choose a more traditional, relational database like PostgreSQL, which offers the best performance in all but one tested case. However, when features such as data retention policies, time bucketing, automatic aggregations are crucial for the developed solution, dedicated time-series databases such as TimescaleDB and InfluxDB become a better choice.

Acknowledgments

The research was supported by the Polish Ministry of Science and Higher Education as a part of the CyPhiS program at the Silesian University of Technology, Gliwice, Poland (Contract No. POWR.03.02.00-00-I007/17-00), by Statuatory Research funds of the Silesian University of Technology, Gliwice, Poland (Grant BKM-576/RAU2/2019 ZAD.1), and partially, by the professorship grant (02/020/RGPL9 /0184) of the Rector of the Silesian University of Technology, Gliwice, Poland.

References

1. Apache Spark (accessed on January 9th, 2020), `https://spark.apache.org/`
2. DBMS popularity broken down by database model (accessed on Februrary 2nd, 2020), `https://db-engines.com/en/ranking_categories`
3. InfluxDB on DB-engines ranking (accessed on Februrary 1st, 2020), `https://db-engines.com/en/system/InfluxDB`

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50426-7_28

4. InfluxDB overview (accessed on Februrary 2nd, 2020), https://www.influxdata.com/products/influxdb-overview/
5. InfluxDB overview (accessed on January 9th, 2020), https://www.influxdata.com/products/influxdb-overview/
6. PostgreSQL documentation (accessed on January 9th, 2020), https://www.postgresql.org/about/
7. PostgreSQL on DB-engines ranking (accessed on Februrary 1st, 2020), https://db-engines.com/en/system/PostgreSQL
8. Raspberry Pi 4 datasheet (accessed on February 4th, 2020), https://www.raspberrypi.org/documentation/hardware/raspberrypi/bcm2711/rpi_DATA_2711_1p0_preliminary.pdf
9. Riak KV documentation (accessed on January 9th, 2020), https://riak.com/products/riak-kv/index.html
10. Riak TS datasheet (accessed on January 9th, 2020), https://riak.com/content/uploads/2016/05/Riak-Riak-TS-Datasheet.pdf
11. Riak TS on DB-engines ranking (accessed on Februrary 1st, 2020), https://db-engines.com/en/system/Riak+TS
12. SQLite documentation (accessed on January 9th, 2020), https://www.sqlite.org/about.html
13. SQLite on DB-engines ranking (accessed on Februrary 1st, 2020), https://db-engines.com/en/system/SQLite
14. TimescaleDB documentation (accessed on January 9th, 2020), https://docs.timescale.com/latest/introduction
15. TimescaleDB on DB-engines ranking (accessed on Februrary 1st, 2020), https://db-engines.com/en/system/TimescaleDB
16. TimescaleDB : SQL made scalable for time-series data (2017), https://pdfs.semanticscholar.org/049a/af11fa98525b663da18f39d5dcc5d345eb9a.pdf
17. Bader, A., Kopp, O., Falkenthal, M.: Survey and comparison of open source time series databases. In: Mitschang, B., Nicklas, D., Leymann, F., Schöning, H., Herschel, M., Teubner, J., Härder, T., Kopp, O., Wieland, M. (eds.) Datenbanksysteme für Business, Technologie und Web (BTW 2017) - Workshopband. pp. 249–268. Gesellschaft für Informatik e.V., Bonn (2017)
18. Bernstein, P.A., Goodman, N.: Concurrency control in distributed database systems. ACM Computing Surveys (CSUR) 13(2), 185–221 (1981)
19. Boule, B.: How to benchmark IoT time-series workloads in a production environment (accessed on January 9th, 2020), https://blog.timescale.com/blog/how-to-benchmark-iot-time-series-workloads-in-a-production-environment/
20. Fadhel, M., Sekerinski, E., Yao, S.: A Comparison of Time Series Databases for Storing Water Quality Data, pp. 302–313 (04 2019)
21. Goldschmidt, T., Jansen, A., Koziolek, H., Doppelhamer, J., Breivold, H.P.: Scalability and robustness of time-series databases for cloud-native monitoring of industrial processes. In: 2014 IEEE 7th International Conference on Cloud Computing. pp. 602–609 (June 2014)
22. Kanchan, K., Gorai, A., Goyal, P.: A review on air quality indexing system. Asian Journal of Atmospheric Environment 9, 101–113 (06 2015)
23. Kiefer, R.: TimescaleDB vs. PostgreSQL for time-series: 20x higher inserts, 2000x faster deletes, 1.2x-14,000x faster queries (accessed on January 9th, 2020), https://blog.timescale.com/blog/timescaledb-vs-6a696248104e/
24. Liu, X., Nielsen, P.: Air quality monitoring system and benchmarking. pp. 459–470 (08 2017)

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50426-7_28

25. Paul, A., Pinjari, H., Hong, W.H., Seo, H., Rho, S.: Fog computing-based IoT for health monitoring system. Journal of Sensors 2018, 1–7 (10 2018)
26. Pungila, C., Fortiş, T.F., Ovidiu, A.: Benchmarking database systems for the requirements of sensor readings. IETE Technical Review 26, 342–349 (08 2009)
27. Sanaboyina, T.P.: Performance Evaluation of Time series Databases based on Energy Consumption. Master's thesis, , Department of Communication Systems (2016)
28. Singh, S.: Optimize cloud computations using edge computing. In: 2017 International Conference on Big Data, IoT and Data Science (BID). pp. 49–53 (Dec 2017)
29. Wlodarczyk, T.W.: Overview of time series storage and processing in a cloud environment. In: 4th IEEE International Conference on Cloud Computing Technology and Science Proceedings. pp. 625–628 (Dec 2012)
30. Yu, S.: ACID properties in distributed databases. Advanced eBusiness Transactions for B2B-Collaborations (2009)

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50426-7_28